Enabling Dynamic Security Management of Networked Systems via Device-Embedded Security

Gregory R. Ganger and David F. Nagle

December 2000
CMU-CS-00-174

Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

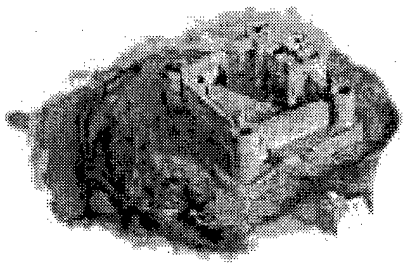
This report contains the technical content of a recent funding proposal. In it, we propose a new approach to network security in which each individual device erects its own security perimeter and defends its own critical resources. Together with conventional border defenses (e.g., firewalls and OS kernels), such self-securing devices *could provide a flexible infrastructure for dynamic prevention, detection, diagnosis, isolation, and repair of successful breaches in borders and device security perimeters.*

Managing network security is difficult in current systems, because a small number of border protections are used to protect a large number of resources. We plan to explore the fundamental principles and practical costs/benefits of embedding security functionality into infrastructural devices, such as network interface cards (NICs), network-attached storage (NAS) devices, video surveillance equipment, and network switches and routers. The report offers several examples of how different devices might be extended with embedded security functionality and outlines some challenge of designing and managing self-securing devices.

Keywords: Security, survivability, intrusion tolerance, storage systems, network-attached storage.

1 Overview

Carnegie Mellon University proposes to develop a new paradigm for network security in which individual devices erect their own security perimeters and defend their own critical resources (e.g., network link or storage media). Together with conventional border defenses, these *self-securing devices* will provide a flexible infrastructure for dynamic prevention, detection, diagnosis, isolation, and repair of successful breaches in borders and device security perimeters. Specifically, this new paradigm will enable:



- the placement of multiple, increasingly-specialized security perimeters between attackers and specific resources,
- independent security perimeters around distinct resources, isolating each from compromises of the others,
- rapid and effective intrusion detection, tracking, diagnosis and recovery, using the still-standing security perimeters as a solid foundation from which to proceed,
- the ability to dynamically shut away compromised systems, throttling their network traffic at its sources and even using secure channels to reactively tell their various internal components (e.g., disks and input/output devices) to increase their protective measures,
- the ability to effectively manage and dynamically update security policies within and among the devices and systems in a network environment.

Current network security practice depends upon the solidity of individual border protections, such as firewalls and COTS operating systems. Unfortunately, this approach is flawed both in theory and practice. Pragmatically, it is difficult to correctly implement and configure such a single border. Worse, once an intruder gets past the border, network security administrators are left with no protections in place – that is, there is no safe haven from which to observe an intruders actions, compartmentalize his effects, and recover to operational status.

Augmenting current border protections with self-securing devices promises much greater flexibility for network security administrators. By having each device (e.g., network card, NAS file server) erect independent security perimeters, the overall network environment has many outposts from which to act when under attack. Devices cannot only protect their own resources, they can observe, log, and react to the actions of other nearby devices. Compromises of one security perimeter will compromise only a small fraction of the network environment, allowing other devices to dynamically identify the problem, warn still-secured devices about the compromised system, raise the security levels of the environment, and so forth.

This change in network security paradigm raises two major research questions, each with a number of sub-questions. First, "what should each device do behind its security perimeter?" Answering this question will require exploration of cost, performance, and flexibility trade-offs, as well as exploring what is possible with the limited information available at any given device. Second, "how does one dynamically manage such a large collection of independent

security perimeters?" Answering this question will require exploration of tools and techniques for marshaling sets of self-securing devices, monitoring their current state, and dynamically updating their policies in the face of changes to and attacks upon the network environment's state.

2 Project Description: Better Security Via Smarter Devices

Managing network security is difficult in current systems, because a small number of border protections (e.g., firewalls and/or host operating systems) are used to protect a large number of resources. For example, an attacker who successfully compromises a host OS gains complete control over all resources of that system, including its peripheral devices. Thus, such an intruder gains the ability to transmit anything onto the network, to modify anything on the disk, and to examine all input device signals (e.g., typing patterns and video feeds). Likewise, an attacker who circumvents firewall-based protection has free reign within the vulnerable network environment. Having shared border protections for large sets of resources creates three fundamental difficulties: (1) the many interfaces and functionalities for the many resources (e.g., consider most multi-purpose host OSs) make correct implementation and administration extremely difficult; the practical implications are daily security alerts for popular COTS OSs (e.g., Windows NT and Linux) and network protocols (e.g., e-mail and web), (2) the ability of successful attackers to freely manipulate everything beyond the border protection greatly complicates most phases of security management, including intrusion detection, diagnosis, and recovery, and (3) having a central point for security checks creates severe performance, fault-tolerance, and flexibility limitations for large-scale environments.

In this proposal, we promote an alternative paradigm in which individual system components erect their own security perimeters and protect their critical resources (e.g., network, storage, or video feed) from intruder tampering. The proposed project will explore the fundamental principles and practical costs/benefits of embedding security functionality into infrastructural devices, such as network interface cards (NICs), network-attached storage (NAS) devices, video surveillance equipment, and network switches and routers. By distributing security functionality amongst physically distinct components, systems can avoid much of the fragility and unmanageability inherent in today's border-based security. Specifically, this new "self-securing devices" paradigm will address the three fundamental difficulties outlined above, by allowing each security perimeter to be simpler (e.g., consider NIC or disk interfaces) by reducing the power that results from compromising just one of the perimeters of the system, and by distributing security enforcement checks among the many components of the system.

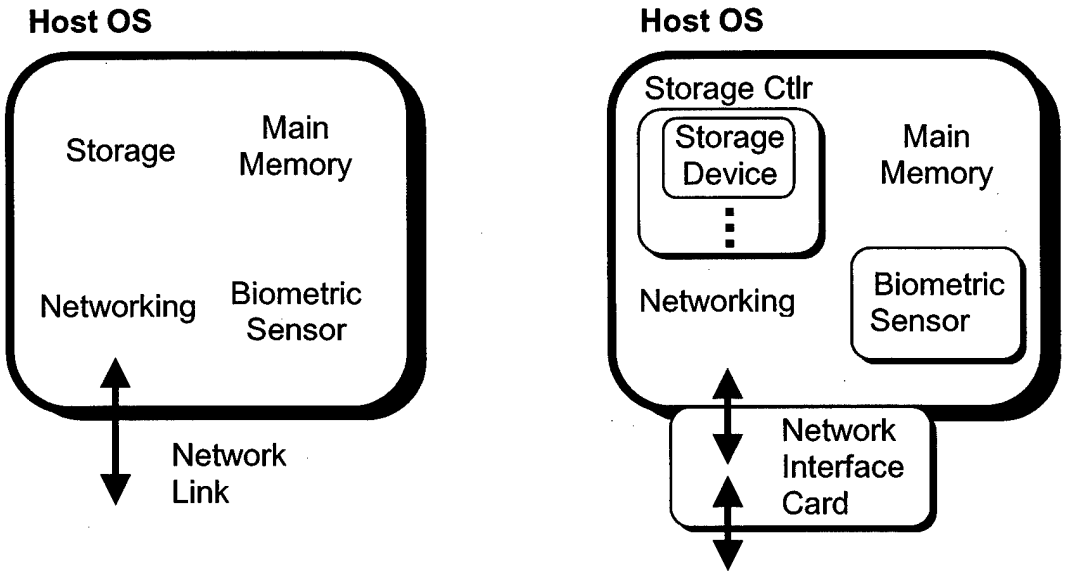
In developing this new paradigm, we will explore the benefits and difficulties involved with embedding security functionality into network interface cards (NICs), storage devices, biometric sensors, network switches and routers, and graphics cards. For example, security-enhanced NICs can provide firewall and proxy server functionality for a given host, as well as throttling or labeling its outbound traffic when necessary. Likewise, self-securing storage devices can protect their data from compromised client systems, and enhanced graphics cards can display warning messages from security administrators even when the window manager is compromised. We will also explore the management approaches and tools needed to integrate all of this new functionality into a robust, flexible defense

from cyber-attacks. For example, we envision graphical tools that combine reports from each security perimeter to show the current state of affairs, as well as dynamic activation tools that allow new policies to be deployed rapidly and directly to specific device perimeters.

The remainder of this proposal describes our inspiration for this paradigm (medieval siege warfare), examples of self-securing devices and how they might be used to effect dynamic network security and some research challenges faced.

3 Siege Warfare in the Internet Age

Despite enormous effort and investment, it has proven nearly impossible to prevent computer security breaches. Together with our growing dependence upon on-line information and wide-area networking, this fact creates an enormous security risk to our national economic and defense infrastructures. To protect our critical information infrastructures, we need defensive strategies that can survive determined and successful attacks, allowing security managers to dynamically detect, diagnose, and recover from breaches in security perimeters. Borrowing from lessons learned in pre-gun warfare, we propose a new network security paradigm analogous to medieval defense constructs.



(a) Conventional OS-based Security

(b) OS + Self-Securing Device

Figure 1: Two security approaches for a computer system. On the left, (a) shows the conventional approach, which is based on a single perimeter around the set of system resources. On the right, (b) shows our new approach, which augments the conventional security perimeter with perimeters around each self-securing device. These additional perimeters offer additional protection and flexibility for defense against attackers. Firewall-enforced network security fits a similar picture, with the new paradigm providing numerous new security perimeters within each system on the internal network.

Current security mechanisms are based largely on singular border protections. This roughly corresponds to defense practices during Roman times, when defenders erected walls around their camps and homes to provide protective cover during attacks. Once inside the walls, however, attackers faced few obstacles to gaining access to all parts of the enclosed area. Likewise, a cracker who successfully compromises a firewall or OS has complete access to the resources protected by these border defenses; no additional obstacles are faced[1]. Of course, border defenses were a large improvement over open camps, but they proved difficult to maintain against determined attackers – border protections can be worn down over time and defenders of large encampments are often spread thin at the outer wall.

As the size and sophistication of attacking forces grew, so did the sophistication of defensive structures. The most impressive such structures, constructed to withstand determined sieges in medieval times, used multiple tiers of defenses. Further, tiers were not strictly hierarchical in nature – rather, some structures could be defended independently of others. This major advancement in defense capabilities provided defenders with significant flexibility in defense strategy, the ability to observe attacker activities, and the ability to force attackers to deal with multiple independent defensive forces.

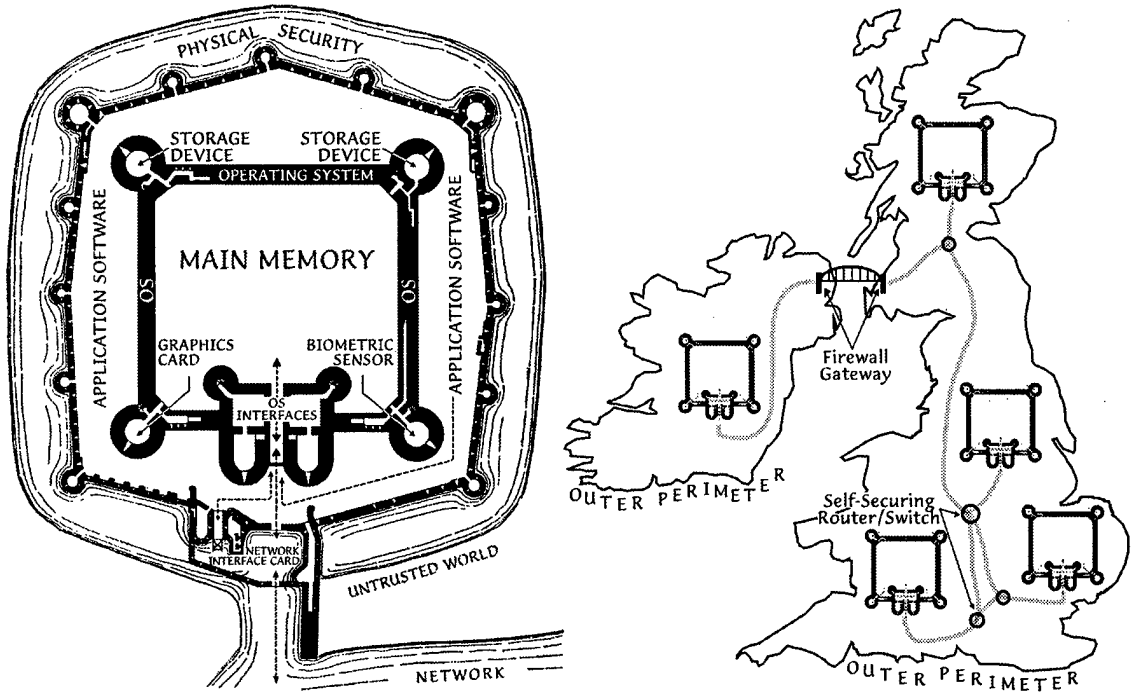
Applying the same ideas to computer and network security, border protections (i.e., firewalls and host OSs) can be augmented with security perimeters erected at many points within the borders. Enabled by low-cost computation (e.g., embedded processors, ASICs), security functionality can be embedded in most device microcontrollers, yielding "better security via smarter devices." We refer to devices with embedded security functionality as *self-securing devices.*

Self-securing devices can significantly increase network security and manageability, enabling capabilities that are difficult or impossible to implement in current systems. For example, independent device-embedded security perimeters guarantee that a penetrated boundary does not compromise the entire system. Uncompromised components continue their security functions even when other system components are compromised. Further, when attackers penetrate one boundary and then attempt to penetrate another, uncompromised components can observe and react to the intruder's attack; from behind their intact security perimeters, they can send alerts to the security administrator, actively quarantine or immobilize the attacker, and wall-off or migrate critical data and resources. Pragmatically, each self-securing device's security perimeter is simpler because of specialization, which should make correct implementations more likely. Further, distributing security checks among many devices reduces their performance impact and allows more checks to be made.

---

[1] This is not quite correct in the case of a firewall protecting a set of hosts that each run a multi-program OS, such as Linux. Such an environment is more like a town of many houses surrounded by a guarded wall. Each house affords some protection beyond that provided by the guarded wall, but not as much in practice as might be hoped. In particular, most people in such an environment will simply open the door when they hear a knock, assuming that the wall keeps out attackers. Worse, in the computer environment, homogeneity among systems results in a single set of keys (attacks) that give access to any house in the town.

By augmenting conventional border protections with self-securing devices, this new security paradigm promises substantial increases in both network security and security manageability. As with medieval fortresses, well-defended systems conforming to this paradigm can survive protracted sieges by organized attackers.



(a) a siege-ready computer system

(b) 2 siege-ready intranets

Figure 2: The self-securing device paradigm illustrated via the siege warfare constructs that inspired it. On the left, (a) shows a siege-ready system with layered and independent tiers of defense enabled by device-embedded security perimeters. On the right, (b) shows two small intranets of such systems, separated by firewall-guarded entry points. Also note the self-securing routers/switches connecting the machines within each intranet.

4 Examples of Device-Embedded Security

To make our new security paradigm more concrete, this section gives several examples of how different devices might be extended with embedded security functionality. In each case, there are difficulties and research questions to be explored; here, we focus mainly on conveying the potential.

Network Interface Cards (NICs): The role of NICs in computer systems is to move packets between the system's components and the network. Thus, the natural security extension is to enforce security policies on packets forwarded in each direction [1]. Like a firewall, a self-securing NIC can do this by examining packet headers and simply not forwarding unacceptable packets into or out of the computer system. A self-securing NIC can also act as a machine-specific gateway proxy, achieving the corresponding protections without the scalability or identification

problems. By performing such functions at each system's NIC, one avoids the bottleneck imposed by current centralized approaches. Further, NIC-based firewalls and proxies will protect systems from insider attacks as well as Internet attacks, since only the one host system is inside the NIC's boundary. Further, self-securing NICs offer a powerful control to network administrators: the ability to throttle or tag network traffic at its sources. So, for example, a host whose security status is questionable could have its network access blocked or limited. Also, security administrators can manage and configure self-securing NICs over the network, since they must obviously be connected directly to it. This allows an administrator to use the NIC to protect the network from its host system. By embedding this traffic management functionality inside the NIC, one enjoys its benefits even when the host OS or other machines inside the LAN border are compromised.

Storage Devices: The role of storage devices in computer systems is to persistently store data. Thus, the natural security extension is to protect stored data from attackers, preventing undetectable tampering and permanent deletion [3]. A self-securing storage device can do this by managing storage space from behind its security perimeter, keeping an audit log of all requests, and keeping previous versions of data modified by attackers. Since a storage device cannot distinguish compromised user accounts from legitimate users, the latter requires keeping all versions of all data. Finite capacities will limit how long such comprehensive versioning can be maintained, but 100% per year storage capacity growth will allow modern disks to keep several weeks of all versions. If intrusion detection mechanisms reveal an intrusion within this multi-week *detection window*, security administrators will have this valuable audit and version information for diagnosis and recovery. This information will simplify diagnosis, as well as detection, by not allowing system audit logs to be doctored, exploit tools to be deleted, or back doors to be hidden – the common steps taken by intruders to disguise their presence. This information will simplify recovery by allowing rapid restoration of pre-intrusion versions and incremental examination of intermediate versions for legitimate updates. By embedding this data protection functionality inside the storage device, one enjoys its benefits even when the network, user accounts, or host OSs are compromised.

Biometric Sensors: The role of biometric sensors in computer systems is to provide input to biometric-enhanced authentication processes, which promise to distinguish between users based on measurements of their physical features. Thus, the natural security extension is to ensure the authenticity of the information provided to these processes. A self-securing sensor can do this by timestamping and digitally signing its sensor information. Such evidence of when and where readings were taken is needed because, unlike passwords, biometrics are not secrets [2]. For example, anyone can lift fingerprints from one's office with the right tools or download facial images from one's web page. Thus, the evidence is needed to prevent straightforward forgery and replay attacks. Powerful self-securing sensors may also be able to increase security and privacy by performing the identity verification step from within their security perimeter and only exposing the results (with the evidence). By embedding mechanisms for demonstrating authenticity and timeliness inside sensor devices, one can verify sensor information (even from over a network) even when intruders gain the ability to offer their own "sensor" data.

Graphical Displays: The role of graphical displays in computer systems is to visually present information to users. Thus, a natural security extension would be to ensure that critical information is displayed. A self-securing display could do this by allowing high-privilege entities to display data that cannot be overwritten or blocked by less-privileged entities. So, for example, a security administrator could display a warning message when there is a problem in the system (e.g., a new e-mail virus that must not be opened). By embedding this screen control inside the display device, one gains the ability to ensure information visibility even when an intruder gains control over the window manager.

Routers and Switches: The role of routers and switches in a network environment is to forward packets from one link to an appropriate next link. Thus, one natural security extension for such devices is to provide firewall and proxy functionality; many current routers provide exactly this. Some routers/switches also enhance security by isolating separate virtual LANs (VLANs). More dynamic defensive actions could provide even more defensive flexibility and strength. For example, the ability to dynamically change VLAN configurations would give security administrators the ability to create protected command and control channels in times of crisis or to quarantine areas suspected of compromise. When under attack, self-securing routers/switches could also initiate transparent replication of data services, greatly reducing the impact of denial-of-service attacks. Further, essential data sites could be replicated on-the-fly to "safe locations" (e.g., write-once storage devices) or immediately isolated via VLANs to ensure security. Self-securing routers/switches can also take an active role in intrusion detection and tracking, by monitoring and mining network traffic. When an attack is suspected, alerts can be sent to administrators and to other self-securing devices to increase security protections. By embedding traffic monitoring and isolation functionality in self-securing routers/switches, one can enjoy its benefits even when firewalls and systems on the internal network are compromised.

5 Examples of Newly-Enabled Dynamic Actions

Many new dynamic network security actions are enabled by the more numerous and heterogeneous security perimeters inherent to the self-securing device paradigm. To illustrate the potential, this section describes a few such actions:

Network DefCon Levels: Often, there is a trade-off between security and performance. For example, the more detailed and numerous the firewall rules, the greater the overhead introduced. Likewise, the more detailed the event logging, the greater the overhead. One use of the many new security perimeters is to support dynamic increases of security level based on network-wide status. For example, if an attack can be detected after only a small number of perimeters are compromised, the security levels at all other self-securing devices can be dynamically raised. As suggested above, this might take the form of more detailed firewalling at NICs, logging of network traffic to storage, and dynamic partitioning of the network into distinct VLANs.

Email Virus Stomping: One commonly observed security problem is the rapidly-disseminated e-mail virus. Even after detecting the existence of a new virus, it often takes a significant amount of time to root it out of systems. Ironically, the common approach to spreading the word about such a virus is via an e-mail message (e.g., "don't open unexpected e-mail that says 'here is the document you wanted' "). By the time a user reads this message, it is often too late. An alternative, enabled by self-securing NICs, is for the system administrator to immediately send a new rule to all NICs: check all in-bound and out-bound e-mail for the new virus's patterns. This would immediately stop further spread of the virus within the intranet, as well as quickly identifying many of the infected systems.

Biometric Identity Verification: A more exotic use of self-securing devices is auxiliary identity checks on users. For example, imagine that an authenticated user does something that triggers an intrusion detection alarm. There are many possible explanations, one of which is that someone else is using the real user's session (e.g., while the real user is away at lunch). To check for this, a network security administrator could silently consult a nearby (or attached) self-securing video camera and perform face or iris recognition. Many other biometrics could also be used. The intrusion detection system could even trigger this check automatically and terminate the corresponding system's network and storage access, if the user is deemed to be an imposter.

Traffic Throttling at the Source: As the previous example suggests, self-securing NICs allow network traffic to be throttled at its sources. Thus, a system that is deemed "bad" could have its network traffic slowed or cut off completely. Also, such malicious network activity as "SYN bombs" and IP address spoofing can be easily detected, terminated at its source, and even automatically repaired (e.g., sending RST packets to clear SYN bomb connections).

Migration of Critical Data from Compromised Systems: If a system is compromised, one important action is trying to save and retain access to user data. In our new paradigm, this can be done by having the self-securing storage device (appropriately and authoritatively directed) encrypt and send the relevant contents over the network via the self-securing NIC. The self-securing router can forward the data to one or more alternate locations and route subsequent accesses to the data appropriately. In fact, different user bases could be routed to distinct replicas (e.g., to isolate command and control access from less-privileged activity). With emerging device-to-device I/O interconnects, the storage-to-network transfer can be done with no host OS involvement at all, leaving the successful intruder with no way to stop it. Going back to the first example, another use of this type of 0support would be to frequently transfer the audit logs from various self-securing devices to on-line intrusion detection systems during perceived siege situations.

Displaying Trojan-defeating Messages: In perhaps the simplest example, a security administrator could direct a self-securing graphics card to override system directives and display a warning message. Such support would be particularly useful when users need to be warned to discontinue (or not start) using a system suspected of housing

Trojan horses. Again, device-to-device communication allows this to happen over the network without host OS interference.

6 Challenge: Design of Self-Securing Devices

For each self-securing device, the research challenge before us is to identify what security functionality should be included and how it should be implemented. Earlier, we outlined potential functionalities for a number of devices. This section goes into more detail for two critical infrastructural devices: network interface cards and storage devices.

6.1 Self-Securing NICs

Self-securing network interface cards (NICS) add a secure perimeter around each host by enforcing security directly at every host network interface [1]. Each NIC contains an embedded firewall that performs both packet-based filtering and more complex application-level proxy filtering. The NIC-based firewall inspects both in- and out-bound traffic, detecting traffic that is either unauthorized (e.g., no FTP send commands) or potentially dangerous (e.g., e-mail viruses). Based on security policies administered by both the network manager and user, suspicious events/data can either be dropped, flagged as potentially dangerous, or logged for future inspection. Self-securing NICs also enable secure mobile computing, maintaining security regardless of a device's location and by protecting machines from wireless devices that may enter their network environment.

The self-securing NIC enforces security policies defined by the network administrator (just like a centralized firewall does today) in addition to security policies defined by a user or host OS. To guarantee that the network-based security policies are enforced, the self-securing NIC is physically partitioned to ensure that the firewall's resources (e.g., memory, computation) are neither compromised nor denied. Physical datapath layouts force network traffic to pass through the NIC firewall before the host receives the data. Even when a host OS is compromised, the NIC's architecture prevents any host OS from bypassing this datapath or preventing the network firewall from filtering the data. Further, because the NIC can filter both incoming and outgoing data, compromised hosts can be prevented from attacking other nodes by having attacks squelched at the source (host) NIC.

Distributing security policy is one of the central research issues in this project. Every NIC will need to coordinate with a trusted security manager responsible for distributing policy. This could be accomplished via trusted network security managers that send policy commands via either secure channels or trust management systems such as Keynote. Because security managers are also susceptible to attack, it is possible that a system of self-securing NICs could be compromised through a compromised security manager. One potential solution would be to disallow a reduction in security policy without a hosts OS's or user's express permission. Therefore, if a manager was compromised, it would not be allowed to reduce the security level of any managed host without permission from that host. Hence, an attacker would have to compromise both the security manager and the host before they could compromise the firewall.

Self-securing NICs also enable a number of optimizations that are difficult or impossible for centralized firewalls. Host-based NICs can utilize specific host-based information, such as the type of application or an application's in-memory data structures to simplify the filtering process. For example, viruses cannot be launched within an audio or video stream. Therefore, the cost of virus checking on these types of data can be avoided. Even in instances where virus checking must occur at the beginning of a data stream (e.g., looking for the first line of a file containing \#!/bin/csh), once a data stream has been deemed secure, that traffic stream can be tagged as secure to avoid the cost of unnecessary filtering. Finally, by placing encryption under the firewall, self-securing NICs can filter data after it has been securely received and decrypted (e.g., with IPsec), but before the application.

Another research area of exploration is the ability of NICs to coordinate their activity, creating ad-hoc realms of trust among multiple NICs. In mobile environments, this could allow each host to share its data within the collective of NICs using one form of security (e.g., encryption) within the collective and another (e.g., full firewall filtering) from outside.

Deploying a fully scalable set of self-securing network interfaces poses numerous fundamental research questions, including:

- What is the best method for managing the security policy? Should centralized realms of trust be used or a more distributed trust management system such as Keynote?
- How can a compromised host or network be detected?
- What is the most efficient architecture for a self-securing network interface card?
- What is the most secure method for downloading new security functionality onto the NIC?
- How can a group of self-securing NICs coordinate activity to maximize protection?
- Can a self-securing NIC leverage a layering of security policies across both centralized and distributed firewalls?
- What happens if a self-securing NIC is itself compromised? Can it be recovered and if so, how?
- How can Self-securing NICs organize to create collectives of trust?

6.2 Self Securing Storage Devices

As described before, self-securing storage devices can protect their data even when the host OS is compromised and the network environment is hostile. From behind its security perimeter, the storage device ensures data survival by keeping previous versions of the data and an audit log of requests. The old versions of objects comprise the *history pool*. Every time an object is modified or deleted, the version that existed just prior to the modification becomes part of the history pool. Eventually an old version will age and have its space reclaimed. A self-securing storage device guarantees a lower bound on the amount of time that a deprecated object remains in the history pool before it is reclaimed. During this window of time, the old version of the object can be completely restored. The guaranteed window of time during which an object can be restored is called the *detection window*, as it indicates the amount of

time that the administrator has to detect the intrusion and have full storage history with which to diagnose and recover. When determining the size of this window, the administrator must examine the tradeoff between the detection latency provided by a large window and the extra disk space that is consumed by the proportionally larger history pool.

Self-securing storage assists in intrusion survival and recovery by allowing the administrator to view audit information and quickly restore modified or deleted files. The audit and version information also helps to diagnose intrusions and detect the propagation of maliciously modified data. For example, self-securing storage simplifies detection of an intrusion since versioned system logs and utility programs cannot be imperceptibly altered. Also, since the administrator has the complete picture of the system's state, from intrusion until discovery, it is considerably easier to establish the method used to gain entry. Previous versions of system files, from before the intrusion, can be quickly and easily restored by resurrecting them from the history pool. Overall, the data protection that self-securing storage provides allows easy detection of modifications, selective recovery of tampered files, prevention of data loss due to out-of-date backups, and speedy recovery since data need not be loaded from an off-line archive.

In a recent paper [3], we proposed self-securing storage, described associated design challenges, and evaluated the feasibility of keeping all versions of all data. Our feasibility study suggests that both the capacity and performance concerns that might be associated with self-securing storage are not show-stoppers. Nonetheless, several research questions remain:

- How does one securely manage self-securing storage devices (e.g., change the detection window), given that attackers could attempt to use almost any path that an administrator can use? (Note that this question recurs for each self-securing device.)
- How does one deal with denial-of-service attacks that attempt to overflow the history pool?
- How does one efficiently encode versions of data and metadata to minimize the history pool size for a given detection window length?
- How can the encoded history information be indexed to make intrusion detection and diagnosis tools more effective?
- What exactly is the performance overhead of maintaining the versions and audit logs?
- How can one utilize the storage device's audit log and version history to detect intrusions? To what extent can this be automated? To what extent can this be made more effective by combination with information from other devices' logs?
- How can one utilize the storage device's audit log and version history to diagnose intrusions (e.g., identify intruder actions)? To what extent can this be made more effective by combination with information from other devices' logs? (For example, can the propagation of tainted information be tracked?)

7 Challenge: Managing and Marshalling Self-Securing Devices

To realize the power of our new security paradigm, approaches and tools must be developed to marshal, monitor, and manage the many security perimeters. Creating dynamic, robust network security environments from large collections of distinct security perimeters will be a major focus of our efforts. This section outlines two powerful tools that we plan to develop and poses some important research questions to be answered.

Visualizing the State of Network Security: We envision a graphical tool that merges status information from the various self-securing devices to display the current state of security within the network. This display could highlight and draw attention to known and suspected trouble spots, helping security administrators rapidly discover and deal with problems. The status information collected from devices can include their state, their attack detection observations, and their observations about devices with which they interact. When concerns arise, the tool could ask devices for more detailed or updated information. For example, when a particular machine's user is behaving oddly, the tool could ask the corresponding biometric sensor about its confidence in the user's authenticity. Similarly, when a client system is behaving oddly, the tool could ask the corresponding storage device for undoctored copies of the audit log for deeper analysis.

Dynamic Network Security Administration: We envision a suite of tools for updating security policies and dealing with problems uncovered by the prior tool. These tools could be integrated into the visualization tool to make responses more efficient. Some example actions include sending more restrictive policies to the self-securing NICs of misbehaving systems, displaying warning messages on self-securing graphical displays, forcing users to re-authenticate with higher confidences, and restoring pre-intrusion versions on a self-securing storage device after rooting out an intruder. As discussed earlier, many other actions are also enabled by self-securing devices.

Effectively creating and utilizing tools of this sort will require much exploration of a number of complex questions, including:

- How does one realize global network security based on the local insights provided by many host systems and self-securing devices?
- How does one achieve robust security when working with perimeters that protect very different resources?
- How should high-level network security policies be translated into local, per-device settings?
- How does one react to the impact of breaches of some security perimeters in this new many-perimeter environment?
- How does one determine the proper reconfiguration steps for isolating compromised components and continuing to operate usefully (if perhaps in a degraded mode)?
- How does one securely administer the many security perimeters and prevent misuse of the administration interfaces?

8 References

[1] David Friedman and David F. Nagle. Building Scalable Firewalls with Intelligent Network Interface Cards CMU-CS-00-173. Carnegie Mellon University School of Computer Science Technical Report, December 2000.

[2] Andrew J. Klosterman and Gregory R. Ganger. Secure Continuous Biometric-Enhanced Authentication. CMU-CS-00-134. Carnegie Mellon University School of Computer Science Technical Report, May 2000.

[3] John. D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A.N. Soules, and Gregory R. Ganger. Self-Securing Storage: Protecting Data in Compromised Systems. Symposium on Operating Systems Design and Implementation. USENIX Association, 2000. San Diego, CA, 23-25 October 2000, pages 165-180.